3 INTERRUPTS AND INTERRUPT SERVICE ROUTINES

3.1 What is an Interrupt?

The dictionary meaning of the word ‘interrupt’ is to break the sequence of operation. While the CPU is executing a program, an interrupt, which comes from some external signal or by a special instruction or condition in the program, breaks the normal sequence of execution of instructions, diverts its execution to some other program called Interrupt Service Routine (ISR) in order to service that interrupt. After executing ISR, the control is transferred back again to the main program which was being executed at the time of interruption.

Suppose you are reading a novel and have completed up to page 100. At this instant, your younger brother asks you to solve his difficulty. You will somehow mark the line and the page you are reading, so that you may be able to continue after you solve his difficulty. Say you have marked page number 101. You will now go to his room to solve his difficulty. While you are helping him a friend of yours comes and asks you for a textbook. Now, there are two options in front of you. The first is to make the friend wait till you complete serving your brother, and thereafter you serve his request. In this, you are giving less priority to your friend. The second option is to ask your brother to wait; remember the solution of the difficulty at the intermediate state; serve the friend; and after the friend is served, continue with the solution that was in the intermediate state. In this case, it may be said that you have given higher priority to your friend. After serving both of them, again you may continue reading from page 101 of the novel. Here, first you are interrupted by your brother. While you are serving your brother, you are again interrupted by a friend. This type of sequence of appearance of interrupts is called nested interrupt, i.e. interrupt within interrupt.

Whenever a number of devices interrupt a CPU at a time, and if the processor is able to handle them properly, it is said to have multiple interrupt processing capability. For example, 8085 has five hardware interrupt pins and it is able to handle the interrupts simultaneously under the control of software. In case of 8086, there are two interrupt pins, viz. NMI and INTR. The NMI is a nonmaskable interrupt input pin which means that any interrupt request at NMI input can not be masked or disabled by any means. The INTR interrupt, however, may be masked using the interrupt flag (IF). The INTR, further, is of 256 types. The INTR types may be from 00 to FFH (or 00 to 255). If more than one type of INTR interrupt occurs at a time, then an external chip called programmable interrupt controller is required to handle them. The same is the case for INTR interrupt input of 8085. Interrupt Service Routines (ISRs) are the programs to be executed by interrupting the main program execution of the CPU, after an interrupt request appears. After the execution of ISR, the main program continues its execution further from the point at which it was interrupted.

3.2 Interrupts Sources in 8086

Broadly, there are two sources of interrupts. The first out of them is external (hardware) interrupt and the second is internal (software) interrupts. The external interrupt occurs when an external device or a signal interrupts the processor from outside or, in other words, the interrupt is generated outside the processor, for example, a keyboard interrupt. The internal interrupt, on the other hand, is generated internally by the processor circuit, or by the execution of an interrupt instruction. The examples of this type are divide by zero interrupt, overflow interrupt, interrupts due to INT instructions, etc.

3.3 Interrupt Service Routines and Interrupt Types

You may remember that when the 8086 does a far call to a procedure, it puts a new value in the code segment register and a new value In the instruction pointer. For an indirect call the 8086 gets the new values for CS and IP from four memory addresses. Likewise, when it responds to an interrupt the 8086 goes to memory locations to get the CS and IF values for the start of the interrupt service procedure. In an 8086 system the first 1 Kbyte of memory from 00000H to 003FFH is set aside as a table for storing the starting addresses of interrupt service procedures. Since 4 bytes are required to store the CS and IP values for each interrupt service procedure, the table can hold the starting addresses for up to 256 Interrupt procedures. The starting address of an interrupt service procedure stored in this table is often called the interrupt vector or the interrupt pointer, and the table itself is then referred to as the interrupt vector table or the interrupt pointer table.

Figure 3.1 shows how the 256 interrupt pointers are arranged in the memory table. Each double word interrupt pointer is identified by a number from 0 to 255. Intel calls this number the type of the interrupt. The lowest five types are dedicated to specific interrupts such as the divide by zero interrupt and the nonmaskable interrupt which we will explain in details later. The next 27 interrupt types, from 5 to 31, are reserved by Intel for use in future microprocessors. The upper 224 interrupt types, from 32 to 255, are available for you to use for hardware or software interrupts.

For obtaining an interrupt address vector, the 8086 uses the two addresses in the pointer table where IP and CS are stored for a particular interrupt type.

For example, for the interrupt type nn (instruction INT nn), the table address for IP = 4 * nn and the table address for CS = (4 * nn) + 2. For servicing the 8086's nonmaskable interrupt (NMI pin), the 8086 assigns the type code 2 to this interrupt. The 8086 automatically execute the INT2 instruction internally to obtain the interrupt address vector as follows:

Address for IP = 4 * 2 = 00008H

Address for CS = (4 * 2) + 2 = 000AH

The 8086 loads the values of IP and CS from the 20-bit physical addresses 00008H and 0000AF in the pointer table. The user must store the desired 16-bit values of IP and CS in these locations. Similarly, the IP and CS values for other interrupts are calculated.

When the 8086 responds to an interrupt, it automatically goes to the specified location in the interrupt pointer table to get the starting address of the interrupt service procedure. The 8086, however, does not automatically load the starting address in the pointer table. As we will show later, user has to do this with instructions at the start of his/her program. Note that the new value for the instruction pointer is put in as the low word of the pointer, and the new value for the code segment register is put in as the high word of the pointer.

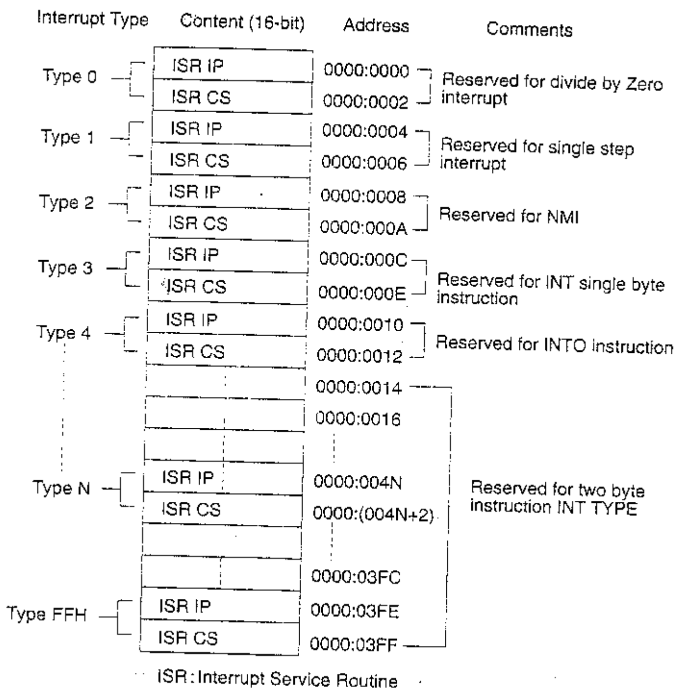


Fig. 3.1: Structure of Interrupt Vector Table of 8086

3.4 <u>Nonmaskable and Maskable Interrupts</u>

The processor 8086/88 has a nonmaskable interrupt input pin (NMI). The NMI is activated on a positive transition (low to high voltage). The assertion of the NMI interrupt is equivalent to an execution of instruction INT 02, i.e. Type 2 INTR interrupt.

The NMI pin should remain high for at least two clock cycles and is not needed to be synchronized with the clock for being sensed. When NMI is activated, the current instruction being executed is completed, and then the NMI is served. Another high going edge on the NMI pin of 8086, during the period, in which the first NMI is served, triggers another response. The signal on the NMI pin must be free of logical bounces to avoid erratic NMI responses.

All software interrupts are nonmaskable interrupts since the processor is going to respond to such interrupts whatever was the status of the Interrupt flag (IF).

In the other hand, maskable interrupts can be sourced to the processor 8086/88 through the pin INTR. This pin can source various interrupt types. The priorities, within the INTR types, are decided by the type of the INTR signal that is to be passed to the processor via data bus by some external device like the programmable interrupt controller. The INTR signal is level triggered and can be masked by resetting the interrupt flag. It is internally synchronized with the high transition of CLK. For the INTR signal, to be responded to in the next instruction cycle, it must go high in the last clock cycle of the current instruction or before that, The INTR requests appearing after the last clock cycle of the current instruction will be responded to after the execution of the next, instruction. The status of the pending interrupts is checked at the end of each instruction cycle.

If the IF is reset, the processor will not serve any interrupt appearing at this pin. If the IF is set, the processor is ready to respond to any INTR interrupt. INTA is only generated by the 8086 in response to INTR signal, telling the external device that it is ready to accept the interrupt and asking for interrupt type to be sent on the lower data bus D7-D0. Once the processor responds to an INTR signal, the IF is automatically reset. If one wants the processor to further respond to any type of INTR signal, the IF should again be set.

A priority interrupt controller such as the 8259A can be used with the 8086 INTR to produce eight levels of hardware interrupts. The 8259A has built-in features for expansion of up to 64 levels with additional 8259As. The 8259A is programmable and can be readily used with 8086 to obtain multiple interrupts from the single 8086 INTR pin.

3.5 Predefined Interrupts (0 to 4)

The predefined interrupts include DIVISION BY ZERO (type 0), SINGLE STEP (type 1) NONMASKABLE INTERRUPT pin (type 2), BREAKPOINT-INTERRUPT (type 3), and INTERRUPT ON OVERFLOW (type 4). The user must provide the desired IP and CS values in the interrupt pointer table. The user may also imitate these interrupts through hardware or software. If a predefined interrupt is not used in a system, the user may assign some other function to the associated type.

The 8086 is automatically interrupted whenever a division by zero is attempted. This interrupt is nonmaskable and is implemented by Intel as part of the execution of the divide instruction. When the TF (TRAP flag) is set by an instruction, the 8086 goes into the single step mode.

Once TF is set to one, the 8086 automatically generates a TYPE 1 interrupt after execution of each instruction. The user can write a service routine at the interrupt address vector to display memory locations and/or register to debug a program. Single step is nonmaskable and cannot be enabled by STI (enable interrupt) or CLI (disable interrupt) instruction.

The nonmaskable interrupt is initiated via the 8086 NMI pin. It is edge triggered (LOW to HIGH) and must be active for two clock cycles to guarantee recognition. It is normally used for catastrophic failures such as power failure. The 8086 obtains the interrupt vector address by automatically executing the INT2 (type 2) instruction internally.

Type 3 interrupt is used for breakpoint and is nonmaskable. The user inserts the one-instruction INT3 into a program by replacing an instruction. Breakpoints are useful program debugging.

The INTERRUPT ON OVERFLOW is a type 4 interrupt. This interrupt occurs if overflow flag (OF) is set and the INTO instruction is executed. The overflow flag is affected, for example, after execution of signed arithmetic such as IMUL (signed multiplication instruction. The user can execute the INTO instruction after the IMUL. If there is an overflow, an error service routine written by the user at the type 4 interrupt address vector is executed, otherwise INTO instruction would pass as NOP.

3.6 <u>Interrupt Response in 8086</u>

Suppose an external device interrupts the CPU at the interrupt pin, either NMI or INTR of 8086, while the CPU is executing an instruction of a program. The CPU first completes the execution of the current instruction. The IP is then incremented to point to the next instruction. The CPU then responses to the requesting device immediately if the signal comes from the NMI pin. If it is an INTR request, the CPU check the IF flag. If the IF is set the interrupt request is acknowledged using the INTA output. If the IF is not set, the interrupt requests are ignored. Note that the responses to the NMI is independent of the IF flag. After interrupt is acknowledged, the CPU computes the vector address from the type of the interrupt that may be passed to the interrupt structure of the CPU internally (in case of software interrupts, NMI, TRAP and Divide by Zero interrupts) & externally:, i.e. from an interrupt

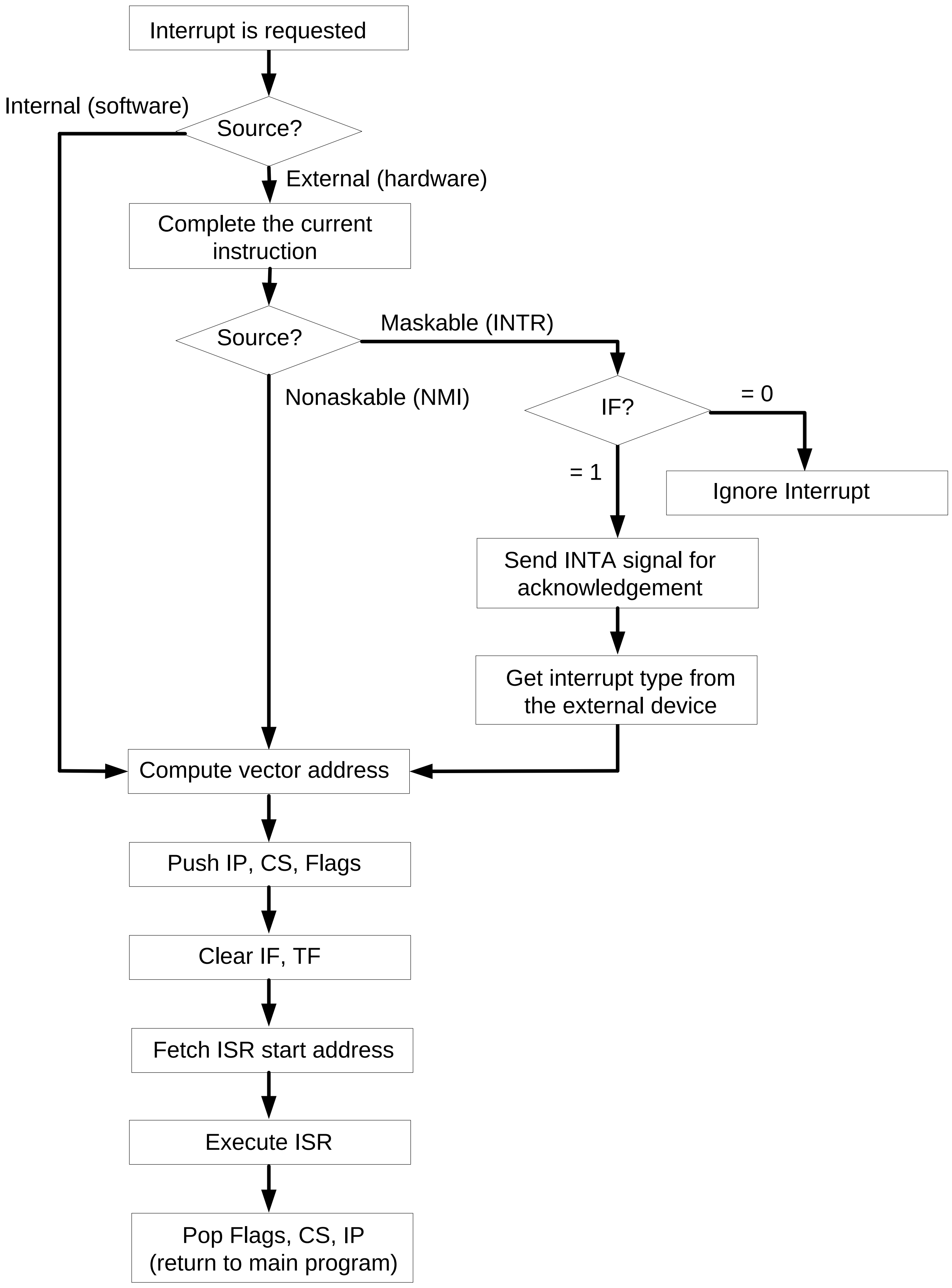


Fig. 3.2: Interrupt Response in 8086

controller in case of external interrupts. The contents of IP and CS are next pushed to the stack. The contents of IP and CS now point to the address of the next instruction of the main program from which the execution is to be continued after executing the ISR. The flag register is also pushed to the stack. The interrupt flag (IF) is cleared. The TF is also cleared. The new address of ISR is found out from the interrupt vector table. The execution of the ISR starts. If further interrupts are to be responded to during the time the first interrupt is being serviced, the IF should again be set to l by the ISR of the first interrupt. If the interrupt flag is not set, the subsequent interrupt signals will not be acknowledged by the processor, till the current one is completed. The programmable interrupt controller is used for managing such multiple interrupts based on their priorities. At the end of ISR the last instruction should be IRET. When the CPU executes IRET, the contents of flags, IP and CS which were saved at the start by the CALL instruction are now retrieved to the respective registers. The execution continues onwards from this address, received by IP and CS. Fig. 3.2 shows how 8086 response to an interrupt request.

3.7 Priority of 8086 interrupts

As you read through the preceding discussions of the different interrupt types, the question may have occurred to you. “What happens if two or more interrupts happen at the same time?’ The answer to this question is that the highest priority interrupt will be serviced first, and then the next highest priority interrupt will be serviced. Table 3.1 shows the priorities of the 8086 interrupts as shown in the Intel data book. Some examples will show you what these priorities actually mean. As a first example, suppose that the INTR input is enabled, the 8086 receives an INTR signal during execution of a divide instruction, and the divide operation produces a divide-by-zero interrupt. Since the internal interrupts such as divide error, INT, and INTO have higher priority than INTR, the 8086 will do a divide error type 01 interrupt response first. Part of the type 0 interrupt response is to clear the IF. This disables the INTR input and prevents the INTR signal from interrupting the higher priority type 0 interrupt service procedure. An IRET instruction at the end of the type 0 procedure will restore the flags to what they were before the type 0 response. This will reenable the INTR input and the 8086 will do an INTR interrupt response. A similar sequence of operations will occur if the 8086 is executing an INT or INTO instruction and a high level signal arrives at the INTR input.

As a second example of how this priority works, suppose that a rising-edge signal arrives at the NMI input while the 8086 is executing a DIV instruction, and that the division operation produces a divide error. Since the 8086 checks for internal interrupts before it checks for an NMI interrupt, the 8086 will push the flags on the stack, clear TF and IF, push the return address on the stack, and go to the start of the divide error (type 0) service procedure. However, because the NMI interrupt request is not disabled, the 8086 will then do an NMI (type 2) interrupt response. In other

Interrupt	Priority
DIVIDE ERROR, INT n, INTO	HIGHEST
NMI	
INTR	
SINGLE-STIP	LOWEST

Table 3.1: Priority of 8086 interrupts.

words, the 8086 will push the flags on the stack, clear TF and IF, push the return address on the stack, and go execute the interrupt service procedure. When the 8086 finishes NMI procedure, it will return to the divide error procedure, finish executing that procedure, and then return to the mainline program.

To finish our discussion of 8086 interrupt priorities let's see how the single step TRAP (or type 1) interrupt fits in. If the trap flag is set, the 8086 will do a type 1 interrupt response after every mainline instruction. When the 8086 responds to any interrupt, however, part of its response is to clear the trap flag. This disables single-step function, so the 8086 will not normally single-step through the instructions of the intend service procedure. In actuality, If the 8086 is in single step mode, when it enters an interrupt service procedure, it will execute the single-step procedure once before it executes the called interrupt procedure. The trap flag can be set again in the single-step procedure if single stepping is desired in the interrupt service procedure.

3.8 Interrupt Programming

While programming for any type of interrupt, the programmer must, either externally or through the program, set the interrupt vector table for that type suitably with the CS and IP addresses of the interrupt service routine. The method of defining the interrupt service routine for software as well as hardware interrupt is the same. Figure 4.7 shows the execution sequence in case of a software interrupt. It is assumed that the interrupt vector table is initialized suitably to point to the interrupt service routine. Figure 3.3 shows the transfer of control for the nested interrupts.

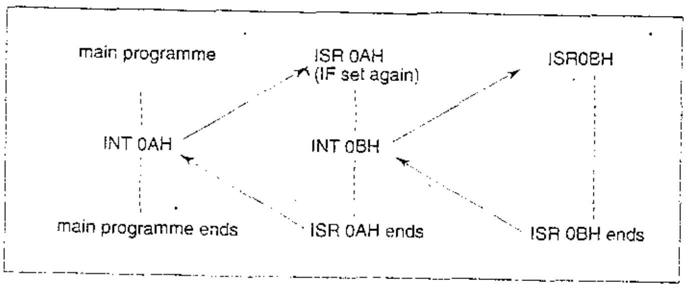
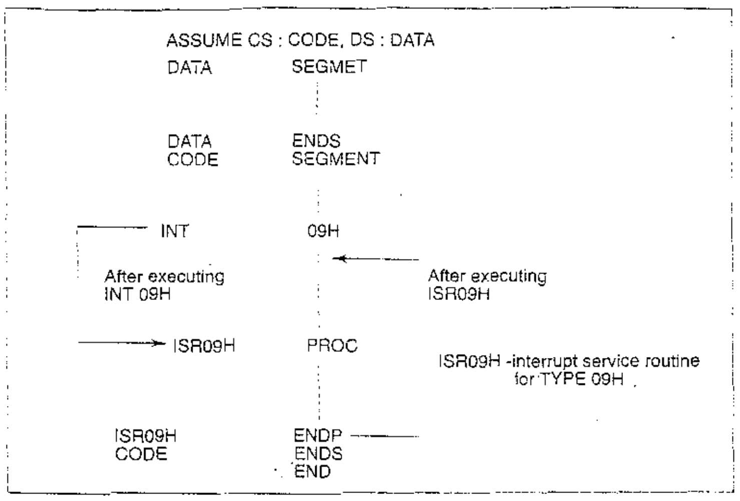


Fig. 3.3: Transfer of Control for Nested Interrupts



```
ASSUME CS : CODE, DS : DATA
DATA          SEGMET
                :
DATA          ENDS
CODE          SEGMENT
                :
        INT   09H
                :
        ISR09H        PROC
                :
        ISR09H        ENDP
        CODE          ENDS
                      END
```

Fig. 3.4: Transfer of Control during Execution of an Interrupt Service Routine

Program 3.1 Write a program to Create a file RESULT and store in it 500H bytes from the memory block starting at 1 000:1 000, if either an interrupt appears at INTR pin with Type 0AH or an instruction equivalent to the above interrupt is executed.

Note: Pin IRQ2 available at IO channel of PC is equivalent to Type 0Ad interrupt.

```
ASSUME    CS:  CODE,       DS:  DATA
DATA       SEGMENT
            FILENAME  DB  "RESULT","$"
            MESSAGE   DB  "FILE WASN'T CREATED SUCCESSFULLY$"
DATA        ENDS
CODE        SEGMENT
START:      MOV    AX, CODE
            MOV    DS, AX                  ;Set DS at CODE for setting IVT.
            MOV    DX, OFFSET  ISR0A       ; Set DX at the offset ot SAGA.
            MOV    AX, 250AH               ;Set IVT using function value
                                           ;2S0AH in AX
            INT    21H                     ; under NT21H.
            MOV    DX, OFFSET  FILENAME    ; Set pointer to Filename,
            MOV    AX,  DATA               ; Set the DS at DATA for Filename
            MOV    DS,  AX                 ; Create file with the File name
                                           'RESULT'.
            MOV    CX,  00
            MOV    AH,  3CH                ; Create file with the File name
                                           ;'RESULT'.
            INT    21H
            JNC    FURTHER                 ;If no carry, create operation is
            MOV    DX,  OFFSET MESSAGE     ; successful else
            MOV    AH,  09H                ; display the MESSAGE.
            INT    21H
            JMP    STOP
```

```
FURTHER     INT     0AH                         ; If the file is created successfully,
            MOV     AH, 4CH                     ; write intc it and return
            INT     21H                         ;to DOS prompt.
                                                ;This interrupt service routine
                                                ;writes 500 bytes into the file
                                                ;RESULT and returns to the main
                                                ;program.


ISR0A       PROC    NEAR
            MOV     BX, AX                      ;Take fife handle in EX,
            MOV     CX, 500H                    ;byte count in CX,
            MOV     DX, 1000H                   ;offset of block in DX,
            MOV     AX, 1000H                   ;Segment value of block
            MOV     DS, AX                      ;in DS.
            MOV     AH, 40H                     ;Write in the file and
            INT     21H                         ;return.
            IRET
ISR0A       ENDP
CODE        ENDS
            END     START
```

To execute the above program, first assemble it using MASM.EXE, link it using LINK.EXE. Then execute the above program at DOS prompt. After execution, you will find a new file RESULT in the directory. Then apply an external pulse to IRQ2 pin of the IBM PC 10 channel. This will again cause the execution of ISR that writes 500 H bytes into the file. For further details of the DOS function calls under INT 21H, refer the MSDOS Encyclopedia or IVIS-DOS Technical Reference.

Program 4.4 Write a program that gives display ‘IRT2 is OK’ if a hardware signal appears on IRQ2 pin and ‘IRT3 is OK’ if it appears on IRQ3 pin of PC 10 Channel.

```
ASSUME     CS:CODE, DS:DATA

DATA        SEGMENT
    MSG1    DB      "IRT2 IS OK", 0AH,
                    0DH,"$"
    MSG2    DB      "IRT3 IS OK", 0AH,
                    0DH,"$"
DATA        ENDS

CODE        SEGMENT
START:      MOV     AX, CODE
```

```
            MOV    DS,   AX                     ;Set IVT for Type 0AH
            MOV    DX,  OFFSET  ISR1
            MOV    AX,  250AH                   ; IRQ2 is equivalent to Type 0AH.
            INT    21H
            MOV    DX,  OFFSET  ISR2            ; Set IVT for Type 0BH.
            MOV    AX,  250BH                   ;IRQ3 is equivalent to TYPE 0BH.
            INT    21H
HERE:       JMP    HERE


                                               ;ISR1 and 1SR2 display the message
ISR1        PROC   LOCAL
            MOV    AX,  DATA
            MOV    DS,  AX
            MOV    DX,  OFFSET  MSG1           ; Display message MSG1.
            MOV    AH,  09H
            INT    21H
            IRET
ISR1        ENDP

ISR2        PROC   LOCAL
            MOV    AX,  DATA
            MOV    DS,  AX
            MOV    DX,  OFFSET MSG2            ; Display message MSG2.
            MOV    AH,  09H
            INT    21H
            IRET
ISR2        ENDP

CODE        ENDS
            END    START
```

Prepare the EXE file of the above program as usual. Execute it at DOS prompt that will hang the system. Now apply a pulse to IRQ9 pin. The message ‘IRT2 is OK’ is displayed on the screen. Then apply a pulse to IRQ pin of 10 channel. The message ‘IRT3 is OK’ is displayed on screen.